Booleans and Conditionals

CS111 Computer Programming
Department of Computer Science
Wellesley College

Booleans

Python has two values of `bool` type, written `True` and `False`. These are called logical values or Boolean values, named after 19th century mathematician George Boole.

The values must be capitalized.

```python
In [1]: True
Out[1]: True

In [2]: type(True)
Out[2]: bool

In [3]: true
NameError: name 'true' is not defined
```

Relational Operators

Booleans most naturally arise in the context of relational operators that compare two values.

```python
In [1]: 3 < 5
Out[1]: ?

In [2]: 3 < 2
Out[2]: ?

In [3]: 3 > 2
Out[3]: ?

In [4]: 5 <= 1
Out[4]: ?

In [5]: 5 >= 1
Out[5]: ?

In [6]: 5 == 5
Out[6]: ?

In [7]: 5 == 6
Out[7]: ?

In [8]: 5 != 6
Out[8]: ?
```

Note `==` is pronounced "equals" and `!=` is pronounced "not equals". This is why we distinguish the pronunciation of the single equal sign `=` as "gets", which is assignment and nothing to do with mathematical equality!
Relational Operators [cont.]

The relational operators can also be used to compare strings (in dictionary order):

```
In [1]: 'bat' < 'cat'
Out[1]: ?
In [2]: 'bat' < 'ant'
Out[2]: ?
In [3]: 'bat' == 'bat'
Out[3]: ?
In [4]: 'bat' < 'bath'
Out[4]: ?
In [5]: 'Cat' < 'bat'
Out[5]: ?
```

In Python (and most other programming languages) uppercase letters come before lowercase letters in string ordering.

Logical Operators in plain English

- **a**: the cake has pineapple  
  - False
- **b**: the cake is chocolate  
  - True
- **c**: the cake has walnuts  
  - True
- **d**: the cake is square  
  - False

**Not**
- **not a**: the cake does not have pineapple  
  - True/False?

**And**
- **a and b**: the cake has pineapple & the cake is chocolate  
  - True/False?
- **b and c**: the cake is chocolate & the cake has walnuts  
  - True/False?

**Or** (slightly different from English...)
- **a or b**: the cake has pineapple or the cake is chocolate  
  - True/False?
- **b or c**: the cake has chocolate or the cake has walnuts  
  - True/False?
- **a or d**: the cake has pineapple or the cake is square  
  - True/False?

In plain English:

- **not exp** evaluates to the opposite of the truth value of **exp**
- **exp1 and exp2** evaluates to True iff both **exp1** and **exp2** evaluate to True.
- **exp1 or exp2** evaluates to True iff at least one of **exp1** or **exp2** evaluates to True.

```
In [1]: not (3 > 5)
Out[1]: ?
In [2]: not (3 == 3)
Out[2]: ?
In [3]: (3 < 5) and ('bat' < 'ant')
Out[3]: ?
In [4]: (3 < 5) and ('bat' < 'cat')
Out[4]: ?
In [5]: (3 > 5) or ('bat' < 'cat')
Out[5]: ?
In [6]: (3 > 5) or ('bat' < 'ant')
Out[6]: ?
```
Truth Tables: **and**

<table>
<thead>
<tr>
<th>exp1</th>
<th>exp2</th>
<th>exp1 and exp2</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>

Truth Tables: **or**

<table>
<thead>
<tr>
<th>exp1</th>
<th>exp2</th>
<th>exp1 or exp2</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>

More Predicates

# determines if n is between lo and hi
def isBetween(n, lo, hi):
    return (lo <= n) and (n <= hi)

def isHogwartsHouse(s):
    return (s == 'Gryffindor' or s == 'Hufflepuff'
    or s == 'Ravenclaw' or s == 'Slytherin')

# determines if n is a prime integer less than 100
def isSmallPrime(n):
    return (isinstance(n, int)
    and (n > 1) and (n < 100)
    and (n == 2 or n == 3 or n == 5 or n == 7
    or not (isDivisibleBy(n,2)
    or isDivisibleBy(n,3)
    or isDivisibleBy(n,5)
    or isDivisibleBy(n,7))))

Predicates

A **predicate** is simply any **function** that returns a boolean value.

# determines if name is Darth Vader
def isDarth(name):
    return name == 'Darth Vader'

# determines whether num is divisible by factor
def isDivisibleBy(num, factor):
    return (num % factor) == 0

# determines whether n is even
def isEven(n):
    return isDivisibleBy(n, 2)

# determines whether strings s1 and s2 have the same length
def sameLength(s1, s2):
    return len(s1) == len(s2)

Define these predicates!

def isFreezing(temp): # temp is at or below freezing °F
def isLongName(name): # name is longer than 20 chars
def isVowel(s): # s is a string that’s a single vowel char

In [2]: isFreezing(10)
Out[2]: True

In [3]: isFreezing(75)
Out [3]: False

In [5]: isLongName('Wellesley')
Out [5]: False

In [4]: isVowel('E')
Out [6]: True

In [7]: isVowel('b')
Out [7]: False
This doesn’t work. Why?

```python
def isVowelBuggy(s):
    letter = s.lower()
    return letter == ('a' or 'e' or 'i' or 'o' or 'u')
```

Making Decisions (revisited)

If it is raining
If three Os in a row/column/diagonal...
If you are rock and your opponent is scissors...
If current block aligns with block below...
If the password matches the one stored, you are logged in
If a comment, then green text, if a string then red text, ...

Conditionals (if Statements)

Booleans are often used to choose between two courses of action in a conditional statement introduced by the keyword `if`.

```python
# returns absolute value of n
def abs(n):
    if n < 0:
        return -n
    else:
        return n
```

Flow Diagrams
Nested Conditionals

```python
if boolean_expression1:
    statement1
    statement2
else:
    if boolean_expression2:
        statement3
        statement4
    else:
        statement5
        statement6
```

A Better Approach: Chained Conditionals

```python
def movieAge(age):
    if age < 8:
        return 'G'
    elif age < 13:
        return 'PG'
    elif age < 18:
        return 'PG-13'
    else:
        return 'R'
```

Flow Diagram: Chained Conditionals

Exercise 1: `letterGrade`

Define a function named `letterGrade` that takes one score (the average of all your individual scores in a class) and returns a letter grade.

Assume:

A >= 90, B >= 80, C >= 70, D >= 60, F < 60
Exercise 2: **addArticle**

Define a function named `addArticle` that takes a string argument and returns a new string with the correct article (a or an) added to the front of the argument.

```python
In [11]: addArticle('cat')
Out[11]: 'a cat'

In [12]: addArticle('ant')
Out[12]: 'an ant'

In [13]: addArticle('')
```

- if `s` is a non-empty string, `s[0]` is a string consisting of the first character in `s`.
- use the `isVowel` function from slide 5-16 as a helper function.

Exercise 3: **daysInMonth**

Define a function named `daysInMonth` that takes a month (as an integer) as the parameter, and returns the number of days in it, assuming the year is not a leap year. If the month does not fall between 1 and 12, return an error message as a string.

```python
In [14]: daysInMonth(3)  # 3rd month is March
Out[14]: 31

In [15]: daysInMonth(9)  # 9th month is September
Out[15]: 30

In [16]: daysInMonth(0)
Out[16]: 'Error! Month should be between 1 and 12'
```

Short-circuit evaluation of **and** and **or**

In `exp1 and exp2 or exp1 or exp2`, the expression `exp2` is not evaluated if the answer is determined by `exp1`.

```python
In[14]: ((1/0) > 0) and (2 > 3)
ZeroDivisionError: integer division or modulo by zero

In[15]: (2 > 3) and ((1/0) > 0)
Out[15]: False

In[16]: (2 < 3) or ((1/0) > 0)
Out[16]: True
```

Combining logical operators

What cake do I like?

```
(cake is chocolate) or (cake has pineapple) and (cake is square)
```

- Parentheses take precedence
- `and` takes precedence over `or` (like * over +)
Creative Predicate Design

There can be many ways of writing the same predicate:

```python
def isVowelOption1(s):
    return (s == 'a' or s == 'e' or s == 'e' or s == 'o'
            or s == 'u' or s == 'A' or s == 'E'
            or s == 'I' or s == 'O' or s == 'U')

def isVowelOption2(s):
    l = s.lower()
    return (l == 'a' or l == 'e' or l == 'e'
            or l == 'o'
            or l == 'u')
```

Long return expressions

It's best to just use parens around long expressions. It is an unexpected but important Python fact that if you want to write long examples without the outermost parens on the return value, you must use the backslash continuation character to end the line (and this character cannot be followed by any other character except newline). Furthermore you must remove internal comments like # Is n an integer?

```python
def isHogwartsHouse(s):
    return s == 'Gryffindor' or s == 'Hufflepuff' \
            or s == 'Ravenclaw' or s == 'Slytherin'
```

All Python values are either Truthy or Falsey

Unexpectedly, in the context of `if`, `and`, and `or`, Python treats a small number of so-called Falsey values (0, '', `None`, [], (), and {}) as False and all other values as True (so-called Truthy values).

In general, we think it is bad style to write code that depends on this fact; use booleans instead!

```python
def testTruthy(val):
    if val:
        return 'Truthy'
    else:
        return 'Falsey'

testTruthy(True) ➞ ‘Truthy’
testTruthy(False) ➞ ‘Falsey’
testTruthy(17) ➞ ‘Truthy’
testTruthy(0) ➞ ‘Falsey’
testTruthy('hello') ➞ ‘Truthy’
testTruthy('') ➞ ‘Falsey’
testTruthy(None) ➞ ‘Falsey’
testTruthy([1, 2, 3]) ➞ ‘Truthy’
testTruthy([]) ➞ ‘Falsey’
```

`isVowelBuggy` revisited

```python
def isVowelBuggy(s):
    letter = s.lower()
    return letter == ('a' or 'e' or 'i' or 'o' or 'u')
```

is equivalent to

```python
def isVowelBuggy(s):
    letter = s.lower()
    return letter == 'a'
```
**Simplifying logical expressions: Distributivity**

A: (cake has pineapple) and (cake has walnuts) or (cake is chocolate) and (cake has walnuts)  
B: cake has walnuts and (cake has pineapple or cake is chocolate)

<table>
<thead>
<tr>
<th>pineapple</th>
<th>chocolate</th>
<th>walnuts</th>
<th>chocolate and walnuts</th>
<th>A</th>
<th>pineapple or chocolate</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>

**Simplifying logical expressions: De Morgan’s Laws**

Law 1.

(cake is not chocolate) and (cake has no walnuts) = not (cake is chocolate or cake has walnuts)

Law 2.

(cake is not chocolate) or (cake has no walnuts) = not (cake is chocolate and cake has walnuts)

**Truth Table for De Morgan’s first Law**

<table>
<thead>
<tr>
<th>chocolate</th>
<th>walnuts</th>
<th>not chocolate</th>
<th>no walnuts</th>
<th>not chocolate and no walnuts</th>
<th>chocolate or walnuts</th>
<th>not (chocolate or walnuts)</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>

Work out truth table for second law at home. 😊